# Interface, exceptions

## Exercice 1 - Evaluateur d'expression II (le retour)

Nous avons lors de la dernière séance représenté les expressions arithmétiques par leur **arbre syntaxique**. Nous avons pour ça défini des classes dont les instances représentent des opérateurs ou des constantes, qui implantent toutes une interface `Expr`, qui déclare une méthode `public double eval()`.
Nous allons maintenant créer des analyseurs syntaxiques dont le rôle est de calculer un tel arbre à partir d'une chaîne de caractères.
Vous documenterez toutes les classes, interfaces et méthodes que vous écrirez en utilisant `javadoc`.
En fait, on souhaite créer deux analyseurs syntaxiques différents, un analyseur préfixe et un analyseur postfixe.
Nous allons donc dans un premier temps essayer de définir un interface (un type) permettant de manipuler indifféremment un analyseur préfixe ou un analyseur postfixe.

1 Écrire une classe d'exception `SyntaxException` qui représente les erreurs syntaxique qui pourront être éventuellement détectée par les analyseurs syntaxiques que nous allons écrire.

2 Écrire une interface `SyntacticAnalyzer` représentant le type des analyseurs syntaxiques. Cette interface doit déclarer une méthode `public Expression analyze(String[] expression) throws SyntaxException`.

## Exercice 2 - Analyseur préfixe

On cherche à construire un évaluateur préfixe respectant l'interface. `SyntacticAnalyzer`.

- Détaillez le fonctionnement de l'analyse préfixe de `* + 1 2 - 3 4`, en construisant au fur et à mesure des opérations l'arbre syntaxique de l'expression.
- Écrire une classe `PrefixAnalyzer` dont la méthode `analyze` prend en paramètre une chaîne de caractères représentant une expression en notation préfixe et implantant l'interface `SyntacticAnalyzer`.

## Exercice 3 - Calculette

Écrire une classe `Calculator` contenant une méthode `main` qui :

- vérifie qu'un paramètre a bien été passée sur la ligne de commandes, et si non appelle une méthode `usage` de la classe `Calculator` qui affiche le mode d'emploi de la calculatrice, et termine ;
- si une chaîne de caractères a bien été passé sur la ligne de commandes, une instance de la classe précédente est créée et est utilisée pour faire l'analyse syntaxique de la chaîne. Vous ferez attention à bien rattraper l'exception qui peut être générée et à l'afficher ;
- si l'analyse syntaxique a réussi, l'expression est évaluée et le résultat est affiché.

## Exercice 4 - Analyseur postfixe

On cherche maintenant à construire l'évaluateur postfixe respectant lui-aussi l'interface. `SyntacticAnalyzer`.

- Détaillez le fonctionnement de l'analyse postfixe de `1 2 + 3 4 * /`, en construisant au fur et à mesure des opérations l'arbre syntaxique de l'expression. Pour faire cela, vous devrez surement utiliser une strcuture de donnée classique. Laquelle est-ce ?
- Écrire une interface pour cette structure de donnée. Vous aurez le soucis d'être le plus général possible dans les profils des méthodes qu'elle déclare.
- Écrire une classe implantant l'interface de la structure de donnée en stockant les éléments dans un conteneur de type `ArrayList`. Donnez la complexité de chacune des méthodes de la classe que vous venez d'écrire.
- Écrire une classe `PostfixAnalyzer` implantant l'interface `SyntacticAnalyzer`. Changer la méthode `main` pour utiliser l'analyseur postfixe.

## Exercice 5 - A la maison : Analyseur postfixe (Suite)

Changer la méthode `main` pour utiliser un analyseur préfixe ou postfix suivant l'option de la ligne de commande et respectant la syntaxe ci-dessous :

```
java Calculator (-prefix|-postfix) expression
```